ED 303 359 SE 050 345

AUTHOR	Gimmestad, Beverly; And Others
TITLE	A Potpourri of Pascal Programs.
SPONS AGENCY	National Science Foundation, Washington, D.C.
PUB DATE	88
GRANT	NSF-DPE-8470653
NOTE	37p.; From a summer workshop entitled "Copper Country Mathematics and Computer Science Teachers Workshop." Drawings may not reproduce well.
PUB TYPE	Guides - Classroom Use - Guides (For Teachers) (052) -- Computer Programs (101)
EDRS PRICE	MF01/PC02 Plus Postage.
DESCRIPTORS	*Computer Graphics; *Computer Software; High Schools; Mathematical Applications; *Mathematical Enrichment; Mathematical Logic; Mathematics Materials; Mathematics Skills; *Mathematics Teachers; *Programing Languages; *Secondary School Mathematics
IDENTIFIERS	IBM PC XT; PASCAL Programing Language

ABSTRACT

This is a collection of Pascal programs that were developed for a 1986 National Science Foundation-sponsored high school teachers' summer workshop. The programs can be used as a means of extending or enriching textbook material in either high school mathematics or Pascal courses. Some suggested uses are: (1) teacher demonstrations in mathematics classes; (2) programs for student use in mathematics classes; and (3) student assignments in a mathematics or Pascal course. The topics which are addressed in the Pascal programs come from a variety of mathematical areas: algebra, trigonometry, discrete mathematics, number theory, mathematical modeling, and numerical algorithms. Generally speaking, the programs are quite short and the programming level is appropriate for the average high school student who is enrolled in a Pascal course or who has completed such a course. The appendix contains standard types and procedures used by graphics programs written for use with Turbo Pascal on the IBM PC, including Cleargraphics, Graphicson, Graphicsoff, Plotpoint, Smoothplot, Drawline, PlotGeneral, DrawlineGeneral, DrawCircleGeneral, plotaxes, ConvertToPolar, and Rotate. (YP)

A POTPOURRI OF PASCAL PROGRAMS

Beverly Gimmestad
Department of Mathematics
Michigan Technological University
Houghton, Michigan 49931

Randolph M. Odendahl
Computer Science Department
State University of New York at Oswego
Oswego, New York 13126

Lynn R. Ziegler
Department of Computer Science
Michigan Technological University
Houghton, Michigan 49931

This material is based upon work supported by the National Science Foundation under Grant No. DPE 8470653. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

The Grant supported a summer workshop entitled "Copper Country Mathematics and Computer Science Teachers Workshop".

TABLE OF CONTENTS

Introduction

A Potpourri of Pascal Programs is a collection of Pascal programs that were developed for an NSF high school teacher's workshop which was held in the summer of 1986. Some of the programs are standard examples in computer science and no source is indicated for these programs. Other programs were taken from published literature and, in these cases, the original source is indicated. Most of the published programs were written in BASIC and translated by us into Pascal. The remaining programs represent original work by either the workshop staff or by the workshop participants. The appropriate author is indicated for these programs. Some of the original programs utilize graphics procedures which are given in the appendix and were written by Dr. Ziegler.

The programs can be used as a means of extending or enriching textbook material in either high school mathematics or Pascal courses. Some suggested uses are:

(1) teacher demonstrations in mathematics classes

(2) programs for student use in mathematics classes

(3) student assignments in a mathematics or Pascal course.

The topics which are addressed in the Pascal programs come from a variety of mathematical areas: algebra, trigonometry, discrete mathematics, number theory, mathematical modeling and numerical algorithms. Generally speaking, the programs are quite short and the programming level is appropriate for the average high school student who is enrolled in a Pascal course or who has completed such a course.

ALGEBRA PROGRAMS

These programs deal with topics in high school algebra courses.

ALGEBRA PROGRAM 1: Binomial Squares.

AUTHOR: Lynn R. Ziegler

OBJECT To dispel an incorrect notion which students often hold about a binomial square.

e.g. $(x+2)^2 \neq x^2 + 4$ but $(x+2)^2 = x^2 + 4x + 4$

Listing of Binomial Squares Program

```
program binomialSquare( input, output );

    var x: integer;

    begin {binomialSquare}
      writeln(lst,' X (X+2)^2 X^2+4 X^2+4X+4');
      writeln(lst,' - ------- ----- --------');
      for x:= -5 to 5 do
         writeln(lst,x:2,sqr(x+2):9,
                          (sqr(x)+4):7,
                     (sqr(x)+4*x+4):10);
    end. {binomialSquare}
```

Sample Output

```
 X    (X+2)^2  X^2+4  X^2+4X+4
 --   -------- -------- -----------
-5       9       29       9
-4       4       20       4
-3       1       13       1
-2       0        8       0
-1       1        5       1
 0       4        4       4
 1       9        5       9
 2      16        8      16
 3      25       13      25
 4      36       20      36
 5      49       29      49
```

ALGEBRA PROGRAM 2: Mixture

AUTHOR: Randy Odendahl (based on an idea in the 1984 NCTM Yearbook, page 187)

Object To compute the cost of a mixture resulting from various proportions of two different coffees. (This is an extension of the problem usually solved in algebra).

Typical problem: If Coffee Type I costs $4.00/lb and Coffee Type II costs $5.40/lb, how many ounces of Coffee I and how many ounces of Coffee II should you combine to make a mixture selling for $4.35/lb?

$$\begin{bmatrix} TypeI \\ x\ oz \end{bmatrix} + \begin{bmatrix} TypeII \\ 16-x\ oz \end{bmatrix} = \begin{bmatrix} Mixture \\ 16\ oz \end{bmatrix}$$

$$\frac{\$4.00}{16}\,x + \frac{\$5.40}{16}\,(16-x) = \frac{\$4.35}{16}\,16$$

$$4.00\,x + 86.40 - 5.40\,x = 69.60$$

$$-1.40\,x = -16.80$$

$$x = \frac{-16.80}{-1.40} = 12\ oz$$

Natural Extension: What would be the cost/lb for various proportions of the two different coffees?

$$4.00\,x + 5.40\,(16-x) = 16\,MIX$$

$$MIX = \frac{4.00\,x + 5.40\,(16-x)}{16}$$

$$MIX = \frac{4}{16}\,x + \frac{5.4}{16}\,(16-x)$$

Listing of Mixture Program

```
program mixture;

{mixture computes cost of mixture resulting from
 various proportions of two different coffees.}

var coffee1: integer;

begin
        writeln(lst,'COFFEE I  COFFEE II  Cost/pound');
        writeln(lst,'--------  ---------  ----------');
        for coffee1:= 0 to 16 do
                writeln(lst,coffee1:5,(16-coffee1):10,
                        ((4.0/16.0)*coffee1+
                        (5.4/16.0)*(16-coffee1)):13:2);
end.
```

Sample Output

```
COFFEE I    COFFEE II    Cost/pound
--------    ---------    ----------
   0           16           5.40
   1           15           5.31
   2           14           5.22
   3           13           5.14
   4           12           5.05
   5           11           4.96
   6           10           4.87
   7            9           4.79
   8            8           4.70
   9            7           4.61
  10            6           4.52
  11            5           4.44
  12            4           4.35
  13            3           4.26
  14            2           4.17
  15            1           4.09
  16            0           4.00
```

ALGEBRA PROGRAM 3: Rectangle

AUTHOR: Bruce Carlson - Dollar Bay High School, Dollar Bay, Michigan 49922

OBJECT Compute the width and area of a rectangle given the perimeter and length.

Listing of Rectangle Program

```
program rectangle(input,output);

{This program is designed to tell a person what the width and area of a
 rectangle would be given the perimeter and length.}

var length,width,perimeter,area:real;

procedure findwidth(perimeter,length:real);
  begin  {findwidth}
    width:=(perimeter/2)-length;
  end; {findwidth}

procedure findarea (length,width:real);
  begin {findarea}
    area:=length*width;
  end; {findarea}

begin {rectangle};
  clrscr;
  writeln ('This exercise is designed to give you the width and the area of ',
        'a rectangle once you have decided upon a perimeter and length.');
  writeln;
  writeln('Try to get the largest area for a given perimeter by changing the length.');
  writeln;
  writeln('What is the perimeter that you would like?) It must be positive.');
  readln (perimeter);
  writeln('What is the length that you would like?');
  readln(length);
  writeln;
  if ((length<(perimeter/2.0)) and (length>0.0)) then begin
    findwidth(perimeter, length);
    writeln('The width of your rectangle is ',width:20:4);
    writeln;
    findarea(length,width);
    writeln('The area of your rectangle is length times width or', area:20:4);
    writeln;
    writeln('Write the perimeter, length, width and area on a sheet of paper.');
    writeln;
    writeln('Try different lengths without changing the perimeter.');
    writeln;
    writeln('What happens to your area as your length and width become',
                    ' closer to each other?');
    writeln;
    writeln('Press the run key (R) before trying a new length.');
    end {if}
    else begin
       writeln('THINK! Your head is not just a hair farm.');
       writeln('Did you choose a POSITIVE PERIMETER?');
       writeln('Also, you must choose a positive length that is less than',
                               ' one-half of the ');
       writeln('perimeter that you choose.  Do you know why?');
       writeln;
       writeln('Now press the run key(R) and try again.');
    end;{if-then-else}
 end.{rectangle}
```

Sample Output

```
This exercise is designed to give you the width and the area of a rectangle once
 you have decided upon a perimeter and length.

Try to get the largest area for a given perimeter by changing thelength.

What is the perimeter that you would like?) It must be positive.
14
What is the length that you would like?
8

THINK! Your head is not just a hair farm.
Did you choose a POSITIVE PERIMETER?
Also, you must choose a positive length that is less than one-half of the
perimeter that you choose.  Do you know why?

Now press the run key(R) and try again.

>r
This exercise is designed to give you the width and the area of a rectangle once
 you have decided upon a perimeter and length.

Try to get the largest area for a given perimeter by changing thelength.

What is the perimeter that you would like?) It must be positive.
14
What is the length that you would like?
4

The width of your rectangle is              3.0000
The area of your rectangle is length times width or          12.0000

Write the perimeter, length, width and area on a sheet of paper.

Try different lengths without changing the perimeter.

What happens to your area as your length and width become closer to each other?


Press the run key (R) before trying a new length.
```

ALGEBRA PROGRAM 4: Shipping

AUTHOR: J. Bugni, L'Anse High School, L'Anse, Michigan 49946

OBJECT Determines if UPS will ship your package given its width, height, and depth.

Listing of Shipping Program

```
{ Shipping program by J. Bugni, L'Anse High School }

program shipping (input,output);

{This propogram determines if UPS will ship your package}

var w,h,d,total:real;

{function upsmeasure finds the sum of the girth and length of a package.}

function upsmeasure (w,h,d:real): real;
{function upsmeasure will find the guth of a package and add to this
 number the length of the package}

var l,g1,g2,g3,girth:real;

begin {upsmeasure}
  l:=w;
  if h>l then l:=h;
  if d>l then l:=d;

  g1:=2*w+2*h;
  g2:=2*h+2*d;
  g3:=2*w+2*d;

  girth:=g1;
  if g2<girth then girth:=g2;
  if g3<girth then girth:=g3;

  upsmeasure:=girth+l;
end; {upsmeasure}

begin {shipping}

  write ('enter the width of your package > ');
  readln (w);
  writeln;
  writeln;
  write ('enter the height of your package > ');
  readln (h);
  writeln;
  writeln;
  write('enter the depth of your package > ');
  readln (d);
```

```
     writeln;
     writeln;
     total:=upsmeasure (w,h,d);
     if total>108 then write ('sorry your package is unacceptable')
     else begin
        write ('we are happy to send your package');
        writeln;
        writeln;
        writeln ('hint:stand your package up so that the longest side is');
        writeln('the vertical side, and put the label on the top!!!');
     end; {else}
  end. {shipping}
```

Sample Output

```
Running
enter the width of your package > 5


enter the height of your package > 12


enter the depth of your package > 9


we are happy to send your package

hint:stand your package up so that the longest side is
the vertical side, and put the label on the top!!!

Running
enter the width of your package > 45


enter the height of your package > 45


enter the depth of your package > 45


sorry your package is unacceptable
```

TRIGONOMETRY PROGRAMS

TRIGONOMETRY PROGRAM 1: Variable Sine Curve

AUTHOR: W. Gaffney, Hancock Public Schools, Hancock, Michigan 49930

OBJECT To change amplitude and period and graphically display a phase shifted sine curve.

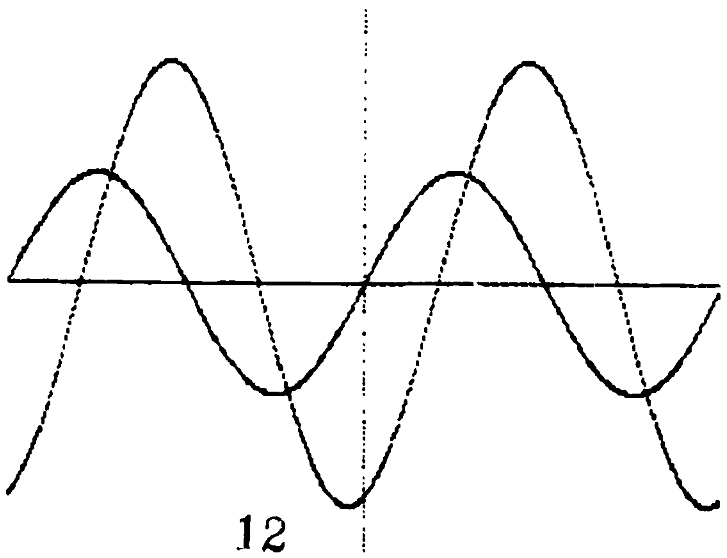
Listing of Variable Sine Curve Program

```
program variablesinecurve;

  {NSF CLASS PROJECT----W. GAFFNEY}
  {This program allows you to change the amplitude and period
   and cause a phase shift of the standard sine curve}

var i,j:integer;
   k,r,a,b,c:real;

   {$igraphics}
   {$igraphics.two}

begin

  writeln('THIS PROGRAM ALLOWS YOU TO CHANGE THE AMPLITUDE',
                ' AND PERIOD');
  writeln('ALONG WITH A PHASE SHIFT OF THE STANDARD',
                ' SINE CURVE');
  writeln('Y=aSIN(bX+c)');
  writeln('THE VALUE OF a WILL DETERMINE THE AMPLITUDE');
  writeln('THE VALUE OF b WILL DETERMINE THE PERIOD');
  writeln('THE VALUE OF c WILL CAUSE A PHASE SHIFT');
  writeln('THE STANDARD CURVE HAS a=1, b=1, c=0 AND IS',
               ' SHOWN IN RED.');

  writeln('INPUT THE VALUE OF a');
  readln(a);
  if (a>2.5) or (a<-2.5) then begin
    writeln('value of a must be in the range of -2.5 to 2.5');
    readln(a);
  end;{if}

  writeln('INPUT THE VALUE OF b');
  readln(b);

  writeln('INPUT THE VALUE OF c');
  readln(c);
```

```
    graphicson;
    cleargraphics;

    {plot the axes}
    for i:=0 to xmax do plotgeneral (i,100,0,319,0,199,1);
    for i:=0 to ymax do plotgeneral (160,i,0,319,0,100,1);

    for i:=0 to 639 do begin
       r:=i*pi/180;

       {plot the standard curve}

       k:=40*sin(1.12*r)+100;
       plotgeneral(i,k,0,639,0,199,2);

       {plot the variable curve}

       k:=a*40*sin((b*1.12*r)+c)+100;
       plotgeneral (i,k,0,639,0,199,3);
    end;
    delay(10000);
    graphicsoff;
 end.
```

Sample Output

```
Running
THIS PROGRAM ALLOWS YOU TO CHANGE THE AMPLITUDE AND PERIOD
ALONG WITH A PHASE SHIFT OF THE STANDARD SINE CURVE
Y=aSIN(bX+c)
THE VALUE OF a WILL DETERMINE THE AMPLITUDE
THE VALUE OF b WILL DETERMINE THE PERIOD
THE VALUE OF c WILL CAUSE A PHASE SHIFT
THE STANDARD CURVE HAS a=1, b=1, c=0 AND IS SHOWN IN RED.
INPUT THE VALUE OF a
2
INPUT THE VALUE OF b
1
INPUT THE VALUE OF c
5
```



DISCRETE MATHEMATICS

DISCRETE MATHEMATICS PROGRAM 1: Factorial and Recursive Factorial.

AUTHOR: Randy Glendahl

OBJECT To contrast the use of an iterative algorithm with the use of a recursive algorithm for generating n!.

Listing of nonrecursive factorial:

```
program factorial(input,output);
var i: integer;
    n: real;
begin {factorial}
  writeln(lst,'What number would you like the factorial of?');
  readln(n);
  write(lst,round(n):1,' factorial is ');
  for i:= round(n) downto 2 do
     n:= n*(i-1);
  writeln(lst,n:20:0);
end. {factorial}
```

Listing of recursive factorial:

```
program recursiveFactorial(input,output);
var
  n:integer;

function factorial( n: integer ): real;
begin {factorial}
  if n=0 then factorial:= 1
  else factorial:= n*factorial(n-1)
end; {factorial}

begin {execute recursiveFactorial}
  writeln(lst,'What number would you like ',
              'the factorial of?');
  readln(n);
  writeln(output,n:1,' factorial is ',factorial(n):20:0);
end. {execute recursiveFactorial}
```

Listing of sample output:

```
What number would you like the factorial of?
17
17 factorial is        355687428096000
```

DISCRETE MATHEMATICS PROGRAM 2: Fibonacci & Recursive Fibonacci

AUTHOR: Randy Odendahl

OBJECT To contrast the use of an iterative algorithm with the use of a recursive algorithm for generating Fibonacci numbers.

Listing of nonrecursive Fibonacci:

```
program Fibonacci;
{Fibonacci prints out the first 20 Fibonacci numbers.}

var  a1,a2,a3,i: integer;
begin {Fibonacci}
  a1:= 1;   a2:= 1;
  writeln(lst,' i   ith Fibonacci');
  writeln(lst,' -   ----------------');
  writeln(lst,1:2,a1:10);
  writeln(lst,2:2,a2:10);
  for i:= 3 to 20 do begin
    a3:= a1 +a2;
    writeln(lst,i:2,a3:10);
    a1:= a2;
    a2:= a3;
  end {for};
end {Fibonacci}.
```

Listing of recursive Fibonacci:

```
program recursiveFibonacci;
var  i: integer;

function Fibonacci( n: integer ): integer;
begin {Fibonacci}
  if (n=1) or (n=2) then Fibonacci:= 1
  else Fibonacci:= Fibonacci(n-1)+Fibonacci(n-2);
end; {Fibonacci}

begin {recursiveFibonacci}
  writeln(lst,' i   ith Fibonacci');
  writeln(lst,' -   ----------------');
  for i:= 1 to 20 do writeln(lst,i:2,Fibonacci(i):10);
end. {recursiveFibonacci}
```

Listing of sample output:

```
i      ith Fibonacci
-      ----------------                   ...
1           1                15         610
2           1                16         987
3           2                17        1597
4           3                18        2584
5           5                19        4181
           ...               20        6765
```

DISCRETE MATHEMATICS PROGRAM 3: Computing powers.

AUTHOR: Lynn R. Ziegler

OBJECT To show a different type of recursive function. (In this case, one useful for computing integer powers of real numbers - x^n. It works by using the observation that $x^{2k} = (x^k)^2$ and $x^{2k+1} = x\ (x^k)^2$.)

Listing of recursivePower:

```
program recursivePower(input,output);

   var x:real;
      n:integer;

   function power(x:real; n:integer) : real;
   var temp:real;
   begin {power}
      if n=0 then power:=1
      else begin
         temp:=power(x,n div 2);
         if (n mod 2) = 0 then power:=sqr(temp)
         else power:=sqr(temp)*x
      end {if then else}
   end; {power}

   begin {recursivePower}
      writeln('Enter x and n ');
      readln(x,n);
      writeln(x:8:2,' to the',
            n:3,'th power is ',
                power(x,n):18:1)
   end. {recursivePower}
```

Sample Output:

```
Enter x and n
2.0 50
   2.00 to the 50th power is 1125899906842620.0
```

NUMBER THEORY

NUMBER THEORY PROGRAM 1: Printmod

AUTHOR: Lynn R. Ziegler

OBJECT To print the first 100 counting numbers modulo the integer input by the user.

Listing of Printmod:

```
program printmod(input,output);
{printmods outputs the first 100 counting numbers modulo the
 integer input by the program's user.}
var i,j: integer;
begin
  writeln(lst,'The first 100 counting numbers ',
                          'in modular arithmetic');
  writeln(lst,'--------------------------------',
                          '----------------------');
  writeln(lst,'Please enter an integer for the modulus:');
  readln(j);
  writeln(lst);
  writeln(lst,'  n ',' n mod ',j:1);
  writeln(lst,'  - ',' --------');
  for i:= 1 to 100 do
    writeln(lst,i:4,(i mod j):4)
end.
```

Sample Output:

```
The first 100 counting numbers in modular arithmetic
------------------------------------------------------
Please enter an integer for the modulus:
6

n  n mod 6
-  ----------                          ...
1      1                         93      3
2      2                         94      4
3      3                         95      5
4      4                         96      0
5      5                         97      1
6      0                         98      2
7      1                         99      3
8      2                        100      4
      ...
```

NUMBER THEORY PROGRAM 2: Wondrous

SOURCE: [Milligan], p. 99 translated to Pascal by Randy Odendahl

OBJECT To determine if a given number is "wondrous". Begin with a whole number. If odd, multiply by 3 and add 1. If even divide by 2. If repeated application of this procedure yields 1 then the original number is called "wondrous". If the given number is not wondrous then the program never halts.

For example, if we start with 102:

```
102/2=61        61*3+1=184      184/2=92     92/2=46          46/2=23
23*3+1=70                          ...
4/2=2           2/2=1  (after 20 steps)
```

Listing of Wondrous

```
program wondrous(input,output);
var     iterationCount,candidate,temp: integer;
        done: boolean;
begin
  done:= false;
  while not done do begin
     writeln(lst,'Please enter number to be tested,',
                         ' or enter 0 to stop.');
     read(candidate);
     if candidate<>0 then begin
       iterationCount:= 0;
       temp:= candidate;
       while temp<>1 do begin
         if odd(temp) then begin
            iterationCount:= iterationCount +1;
            temp:=          temp*3 +1;
         end {if odd};
         iterationCount:= iterationCount +1;
         temp:=          temp div 2;
       end {while};
       writeln(lst,iterationCount,' iterations were ',
                 'needed to make ',candidate,' wondrous.');
       end {if candidate}
     else done:= true;
  end {while not done};
end.
```

Listing of sample output:

```
Please enter number to be tested or enter 0 to stop.
102
20 iterations were needed to make 102 wondrous.
Please enter number to be tested or enter 0 to stop.
0
```

MATHEMATICAL MODELLING

MATHEMATICAL MODELLING PROGRAM 1: ExpGrowth

AUTHORS: Beverly Gimmestad and Randy Odendahl

OBJECT Model growth of the US population from 1790-1970 using the exponential function $P(t) = 3.929\ e^{.029655t}$. Population estimates are printed at ten-year intervals.

Listing of ExpGrowth:

```
program expGrowth(input,output);

const   baseYear=   1790;
        stopYear=   1970;
var     elapsedTime : integer;
begin
  writeln(lst,'YEAR   POPULATION(millions)');
  elapsedTime:= 0;
  repeat
    writeln(lst,baseYear+elapsedTime:4,
              3.929*exp(0.029655*elapsedTime):15:3);
    elapsedTime:= elapsedTime +10;
  until ( (baseYear +elapsedTime) > stopYear );
end.
```

Sample Output:

```
YEAR    POPULATION(millions)
1790          3.929
1800          5.285
1810          7.110
1820          9.564
1830         12.866
1840         17.307
1850         23.282
1860         31.319
1870         42.131
1880         56.675
1890         76.240
1900        102.559
1910        137.963
1920        185.589
1930        249.656
1940        335.840
1950        451.775
1960        607.733
1970        817.528
```

MATHEMATICAL MODELLING PROGRAM 2: Prey-Predator

SOURCE: [Wapner], pp. 137-8 translated into Turbo Pascal by Randy Odendahl.

OBJECT Present a mathematical model describing the population of rabbits (r) and the population of wolves (w) using difference equations. (See **The Mathematics Teacher**, February 1984, 137-138.)

Note: For the differential equaions approach see Kemeny and Snell's **Mathematical Models in the Social Sciences**. Blaisdell Publishing Co, 1962.

In the absence of wolves the population of rabbbits would grow at a rate proportional to its size ($\Delta r = a\ r,\ a > 0$). In the absence of rabbits, the population of wolves would die at a rate proportional to its size ($\Delta w = -d\ w,\ d > 0$). When the two populations interact, the population of rabbits will be decreased by a term proportional to the number of kills and the population of wolves will be increased by a term proportional to the number of kills. The number of kills will vary jointly as r and w.

$$\begin{aligned} \Delta r &= a\ r\ -\ b\ r\ w \\ \Delta w &= c\ r\ w\ -\ d\ w \quad \textit{where}\ \ a, b, c, d\ > 0 \end{aligned}$$

Listing of preypredator:

```
program preypredator(input,output);

 {delta r = ar-brw, delta w= crw-dw.
  the values of a,b,c,and d given in the data
  statement below will give an equilibrium
  point at 300 rabbits and 200 wolves}

 const  a=0.04; b=0.0002; c=0.0001; d=0.03;
 var    r,w:real;
        t: integer;
 begin {preypredator}
   writeln(lst,'Enter initial number of rabbits',
              ' and wolves');
   read(input,r,w);
   writeln(lst,' 25 year period rabbits wolves');
   writeln(lst);
   writeln(lst,round(r):7, round(w):7);
   for t:= 1 to 1000 do begin
     r:= r +(a*r -b*r*w);
     w:= w + (c*r*w -d*w);
     if t mod 25 = 0 then
       writeln(lst,(t div 25):8,round(r):12,round(w):7);
     end;
 end. {preypredator}
```

Sample Output:

```
Enter initial number of rabbits and wolves
350 120
 25 year period  rabbits  wolves                   ...
            1       350     120          22        211       130
            2       482     163          23        313       117
            3       464     262          24        451       145
            4       284     314          25        494       232
            5       179     256          26        333       313
            6       162     183          27        197       278
            7       201     134          28        161       201
            8       294     117          29        185       145
            9       431     137          30        263       119
           10       502     216          31        392       127
           11       362     308          32        502       188
           12       210     288          33        415       290
           13       162     212          34        241       304
           14       178     151          35        167       233
           15       248     120          36        168       165
           16       371     123          37        223       126
           17       494     174          38        332       118
           18       441     277          39        469       154
           19       261     310          40        481       248
           20       172     245          41        306       314
           21       165     173
                    ...
```

MATHEMATICAL MODELING PROGRAM 3: Buffalo Simulation

SOURCE: Dwayne Channeell and Christian Hirsch, "Computer Methods for Problem Solving in Secondary School Mathematics", 1984 SCTM Yearbook, pp. 178-181. (See Bibliography for a full reference for the yearbook.)

Translated to Pascal by Randy Odendahl.

OBJECT The object of this program is to develop a mathematical equation which can predict buffalo herd size over the next ten years, based upon knowledge of the current number of adult males, adult females, male calves, and female calves.

Note: It was desired to have a harvesting policy which would not actually endanger the buffalo population and this program was developed to assist in the formulation of such a policy.

Listing of Buffalo Simulation

```
program BuffaloSimulation(input,output);
var
  {SIMULATION VARIABLES}
  adultMales,adultFemales,maleCalves,femaleCalves,
  babyMales,babyFemales,yearlingMales,yearlingFemales,
  herdSize: real;

  year: integer;

  {TEMPORARY STORAGE DURING CALCULATIONS}
  tAdultMales,tAdultFemales,tBabyMales,tBabyFemales,
  tYearlingMales,tYearlingFemales: real;

  {FORMATTING VARIABLES}
  i: integer;

begin
  writeln(lst,'Enter number of adult males');
  readln(adultMales);
  writeln(lst,round(adultMales):1);
  writeln(lst,'Enter number of adult females');
  readln(adultFemales);
  writeln(lst,round(adultFemales):1);
  writeln(lst,'Enter number of male calves');
  readln(maleCalves'
  writeln(lst,round(maleCalves):1);
  writeln(lst,'Enter number of female calves');
  readln(femaleCalves);
  writeln(lst,round(femaleCalves):1);

  {MODEL ASSUMES TWO-THIRDS OF CALVES ARE NEWBORN
   ONE-THIRD 1 YEAR OLD}
  babyMales:=      2.0/3.0*maleCalves;
  yearlingMales:=  maleCalves-babyMales;
  babyFemales:=    2.0/3.0*femaleCalves;
  yearlingFemales:= femaleCalves-babyFemales;

  writeln(lst); writeln(lst,'          BUFFALO POPULATION DISTRIBUTION');
  writeln(lst,'HERD':6,'ADULT':10,'ADULT':10,'MALE':10,'FEMALE':10);
  writeln(lst,'SIZE':6,'MALES':10,'FEMALES':10,'CALVES':10,'CALVES':10);
  for i:= 1 to 50 do
    write(lst,'-');
  writeln(lst);
```

```
    {BEGIN SIMULATION}
    for year:= 1 to 11 do begin
      {COMPUTE HERD DISTRIBUTION FOR EACH YEAR}
      herdSize:= adultMales+adultFemales+babyMales+babyFemales+
            yearlingMales+yearlingFemales;
      writeln(lst,round(herdSize):6,
            round(adultMales):10,   round(adultFemales):10,
            round(babyMales+yearlingMales):10,
            round(babyFemales+yearlingFemales):10);

      {CALCULATE NEW DENSITIES}
      tAdultMales:=       adultMales;
      tAdultFemales:=     adultFemales;
      tBabyMales:=        babyMales;
      tBabyFemales:=      babyFemales;
      tYearlingMales:=    yearlingMales;
      tYearlingFemales:=  yearlingFemales;
      adultMales:=        0.9*tAdultMales +0.6*tYearlingMales -1000;
      adultFemales:=      0.9*tAdultFemales +0.6*tYearlingFemales;
      babyMales:=         0.48*tAdultFemales;
      babyFemales:=       0.42*tAdultFemales;
      yearlingMales:=     0.5*tBabyMales;
      yearlingFemales:=   0.5*tBabyFemales;
      end {for};
  end. {program buffaloSimulation}
```

Sample output:

```
Enter number of adult males
10400
Enter number of adult females
9100
Enter number of male calves
3380
Enter number of female calves
3120

 BUFFALO POPULATION DISTRIBUTION
HERD   ADULT   ADULT    MALE   FEMALE
SIZE   MALES  FEMALES  CALVES  CALVES
------------------------------------------
26000  10400    9100    3380    3120
28207   9036    8814    5495    4862
28393   7808    8557    6415    5613
27853   7338    8848    6223    5445
27760   6873    9073    6300    5513
27810   6418    9244    6479    5669
27888   6050    9435    6615    5788
28037   5752    9634    6747    5904
28260   5508    9836    6889    6028
28544   5316   10041    7033    6154
28885   5171   10251    7180    6283
```

MATHEMACIAL MODELLING PROGRAM 4: Falling Bodies

AUTHOR: Ernest Mattson, Ironwood Area Schools, Ironwood, Michigan 49938

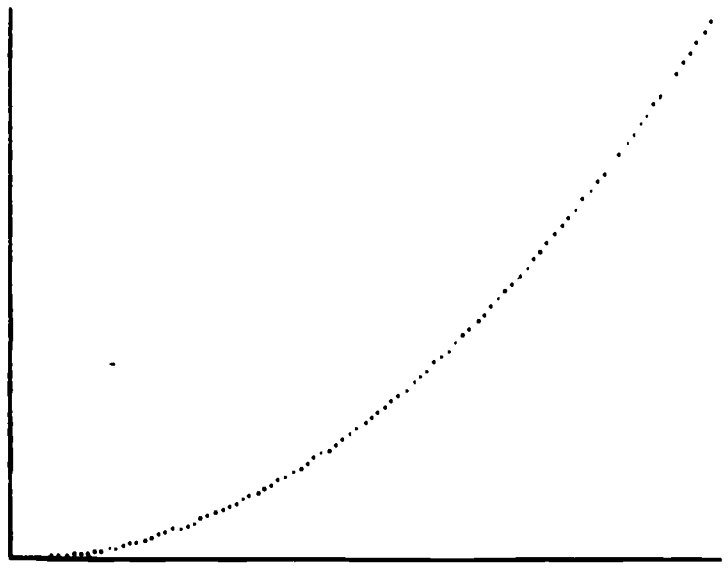
OBJECT Model the behavior of falling bodies according to Newton's law.

Listing of Falling Bodies

```
PROGRAM FALLINGBODIES (INPUT,OUTPUT);

{THIS PROGRAM WILL FIND THE DISTANCE THAT A FREELY
 FALLING BODY  WILL FALL IN THE TIME PERIOD YOU
 SPECIFY.  IT WILL MAKE A CHART OF THE COMPUTED
 VALUES AND DRAW THEIR GRAPH.  YOU MUST ENTER THE
 BEGINNING AND ENDING TIMES AS REAL NUMBERS EQUAL
 TO OR GREATER THAN ZERO.   WHEN YOU ENTER A
 BEGINNING NUMBER GREATER THAN ZERO THE COMPUTED
 VALUES AND THE GRAPH WILL SHOW THE INTERVAL OF TIME
 THAT YOU ENTERED AND THE CORRESPONDING DISTANCES.}

CONST G=9.8;
VAR S:REAL; T:REAL; T2:REAL;
   LAST:REAL;
   DELTAT:REAL;

{$IGRAPHICS.PAS}
{$IGRAPHICS.TWO}

PROCEDURE PRCDRW (X0,Y0,X1,Y1,COLOR:INTEGER);

VAR SLOPE:REAL;
   X,Y:INTEGER;

BEGIN
  IF (X1<>X0) THEN BEGIN
    SLOPE:=(Y1-Y0)/(X1-X0);
    IF (X1>X0)THEN BEGIN
      FOR X:=X1 DOWNTO X0 DO BEGIN
         PLOTPOINT (X,ROUND(SLOPE*(X-X1)+Y1),COLOR);
      END; (*FOR*)
      END (*IF*)
    ELSE IF (X0>X1)THEN BEGIN
      FOR X:=X0 DOWNTO X1 DO BEGIN
         PLOTPOINT (X,ROUND (SLOPE*(X-X0)+Y0),COLOR);
      END; (*FOR*)
    END; (*ELSE*)
    END (*IF*)
  ELSE IF (Y0>Y1) THEN BEGIN
    FOR Y:=Y0 DOWNTO Y1 DO BEGIN
       PLOTPOINT (X0,Y,COLOR);
    END (*FOR*)
    END (*IF*)
  ELSE BEGIN
    FOR Y:=Y1 DOWNTO Y0 DO BEGIN
       PLOTPOINT (X0,Y,COLOR);
    END (*FOR*)
  END(*IF THEN ELSE*)
END;

BEGIN

  WRITELN(OUTPUT,'ENTER THE BEGINNING TIME.');
  WRITELN(OUTPUT,' IT MUST BE A REAL NUMBER GREATER THAN',
                'OR EQUAL TO ZERO.');
  READLN (T2);
  WRITELN(OUTPUT,'ENTER THE TOTAL TIME THAT THE OBJECT',
                   ' FALLS.');
  WRITELN(OUTPUT,'THIS VALUE MUST BE GREATER THAN THE',
             ' BEGINNING TIME.'):
  READLN (LAST);
  WRITELN(OUTPUT,'ENTER THE INTERVAL YOU WANT THE TIME',
                  ' TO BE DIVIDED BY.');
  WRITELN(OUTPUT,'USE A DECIMAL TO REPRESENT A FRACTIONAL',
             ' PART OF A SECOND.');
  READLN (DELTAT);
  WRITELN(OUTPUT,'TIME    DISTANCE');
  T:=T2;
  WHILE (T<=LAST) DO BEGIN
     S:=(G*T*T)/2;
     WRITELN(OUTPUT,T:10:4,S:10:4);
     T:=T+DELTAT;
  END;(*WHILE*)
  READLN;
  WRITELN(OUTPUT,'IF YOU WOULD PLEASE WAIT I ',
             'WILL TRY TO DRAW A GRAPH OF THESE VALUES.');
  DELAY(2000);
  T:=T2;
  GRAPHICSON;
  CLEARGRAPHICS;
  PRCDRW(0,0,XMAX,0,1);
  PRCDRW(0,0,0,YMAX,1);
  WHILE(T<=LAST) DO BEGIN
     S:=(G*T*T)/2;
     PLOTGENERAL (T,S,0.0,LAST,0.0,(G*LAST*LAST)/2,1);
     T:=T+DELTAT;
  END;(*WHILE*)
  READLN;
  GRAPHICSOFF;
END.
```

Sample Output

```
Running
ENTER THE BEGINNING TIME.
IT MUST BE A REAL NUMBER GREATER THAN OR EQUAL TO ZERO
0.0
ENTER THE TOTAL TIME THAT THE OBJECT FALLS.
THIS VALUE MUST BE GREATER THAN THE BEGINNING TIME.
1.0
ENTER THE INTERVAL YOU WANT THE TIME TO BE DIVIDED BY.
USE A DECIMAL TO REPRESENT A FRACTIONAL PART OF A SECOND.
0.01
TIME   DISTANCE
  0.0000  0.0000
  0.0100  0.0005
  0.0200  0.0020
  0.0300  0.0044
  0.0400  0.0078
  0.0500  0.0123
  0.0600  0.0176
  0.0700  0.0240
  0.0800  0.0314
  0.0900  0.0397
  0.1000  0.0490
  .
  .
  .
  0.9000  3.9690
  0.9100  4.0577
  0.9200  4.1474
  0.9300  4.2380
  0.9400  4.3296
  0.9500  4.4223
  0.9600  4.5158
  0.9700  4.6104
  0.9800  4.7060
  0.9900  4.8025
```



NUMERICAL PROGRAMS

NUMERICAL PROGRAM 1: Monte Carlo Area

AUTHOR: Randy Odendahl

OBJECT To use a Monte Carlo method to estimate the area of a circle. This is done by using the monitor screen (size 320 by 200 = 64000 square pixels) and a circle of radius 100 on the screen. x is chosen uniformly between 0 and 319 and y is chosen uniformly between 0 and 199. Then if we repeat this count 20000 times we see that

$$\frac{\textit{Area of circle}}{\textit{Area of screen}} \approx \frac{\textit{Dots in circle}}{\textit{TotalDots}} = \frac{\textit{Dots in circle}}{20000}$$

$$\textit{Area of circle} \approx \frac{\textit{Dots in circle}}{20000} * \textit{Area of screen}$$

$$\textit{Area of circle} \approx \frac{\textit{Dots in circle}}{20000} * 64000$$

$$\textit{Therefore, Area of circle} \approx 3.2 * \textit{Dots in circle}$$

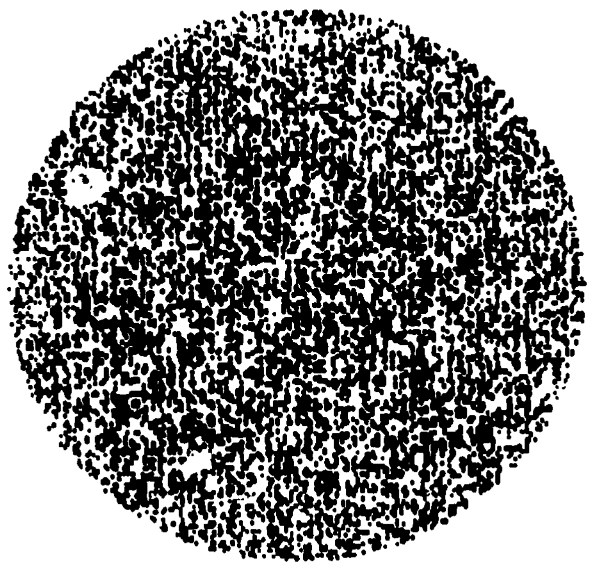
Listing of monteCarloArea:

```
program monteCarloArea;
{monteCarloArea uses Monte Carlo methods
 to estimate the area of a circle.}

 {$Ib:graphics.pas}  {See appendix for a listing of graphics.pas}

const
   xcenter=   320;
   ycenter=   100;
   radius=    100;
   totalCount= 20000;
var
   x,y:         real;
   insideCount: integer;
   i:           integer;

begin
   randomize;  clearGraphics;  graphicsOn;
   insideCount:= 0;

   for i:= 1 to totalCount do begin
      x:= random(xmax);
      y:= random(ymax);
      if (( sqr(x-xcenter)+sqr(y-ycenter))
           <= sqr(radius) ) then begin
         insideCount:= insideCount +1;
         plotPoint(round(x),round(y),1);
      end {if};
   end {for};
```

```
  delay(10000);
  graphicsOff;
  writeln(output,'There were ',insideCount:1,
              ' points inside the circle and');
  writeln(output,'         ',totalCount-insideCount):1,
              ' points outside');
  writeln(output,'The estimated area of the circle',
           ' is ',( (1.0*insideCount/totalCount)*
           (1.0*xmax*ymax) ):1:2);
  writeln(output,'The actual area is ',pi*sqr(radius):1:2);
end.
```

Listing of sample output:

```
There were 10028 points inside the circle
              and 9972 points outside.
The estimated area of the circle is 32089.60
The actual area is 31415.93
```



NUMERICAL PROGRAM 2: Minimize area.

SOURCE: Dwayne Channel and Christian Hirsch, "Computer Methods for Problem Solving in Secondary School Mathematics", 1984 NCTM Yearbook, pp. 174-175. (See Bibliography for a full reference for the Yearbook.)

Translated into Pascal by Randy Odendahl.

OBJECT To investigate the optimal radius and height of a cylindrical container of fixed volume to minimize its surface area.

$$V = \pi r^2 h$$

Write h and Surface [illegible] as functions of V and r.

$$h = \frac{V}{\pi r^2}$$

$$S = 2\pi r h + 2\pi r^2 = 2\pi r(h + r)$$

Listing of minimizeArea:

```
program minimizeArea(input,output);
const pi=3.1415926;

var radius, height, surfaceArea, deltaRadius, volume: real;
   totalTrials, trial: integer;

begin {minimizeArea}
  volume:=236.0; radius:=0.5;  deltaRadius:=0.5; totalTrials:=20;
  writeln(output,'Program to investigate optimal dimensions to minimize ');
  writeln(output,'surface area of a cylindrical container of fixed volume ');
  writeln;
  writeln(output,'Enter fixed volume of the cylinder.');
  writeln(output,volume:5:1);
  writeln(output,'Enter initial length of radius.');
  writeln(output,radius:3:1);
  writeln(output,'Enter increment for radius.');
  writeln(output,deltaRadius:3:1);
  writeln(output,'Enter number of trials to investigate.');
  writeln(output,totalTrials);
  writeln(output,'VOLUME = ',volume:1:2,' TRIALS =',totalTrials:1);
  writeln(output,'   RADIUS    HEIGHT   SURFACE AREA');
  writeln(output,'   ------    ------   ------------');

  for trial:= 1 to totalTrials do begin
    height:= volume/(pi*sqr(radius));
    surfaceArea:= 2*pi*radius*(height + radius);
    writeln(output,radius:10:2,height:10:2,surfaceArea:15:2);
    radius:= radius +deltaRadius;
    end {for}
end. {minimizeArea}
```

Sample output:

```
Program to investigate optimal dimensions to minimize
surface area of a cylindrical container of fixed volume.

Enter fixed volume of the cylinder.
236.0
Enter initial length of radius.
0.5
Enter increment for radius.
1.0
Enter number of trials to investigate.
10
VOLUME = 236.00 TRIALS =20
 RADIUS  HEIGHT  SURFACE AREA
 ------  ------  ------------
   0.50  300.48     945.57
   1.50   33.39     328.80
   2.50   12.02     228.07
   3.50    6.13     211.83
   4.50    3.71     232.12
   5.50    2.48     275.88
   6.50    1.78     338.08
   7.50    1.34     416.36
   8.50    1.04     509.49
   9.50    0.83     616.74
```

NUMERICAL PROGRAM 3: Bisection

AUTHOR: Lynn R. Ziegler

OBJECT The following program is a graphics program to graphically show how the method of bisection can be used to find a zero of a function. In our case the function is $y=2-e^x$ in the range $0 \le x \le 1$. We expect to find the root at ln(2)=0.6931472... .

Bisection works by evaluating the function at the ends of an interval where the function has different signs (positive at one end, negative at the other). The function is then evaluated at the midpoint of that interval. If the function is zero the root is found; if positive, the new interval will be between the midpoint and the end having a negative functional value; if negative, the new interval will be between the midpoint and the end having a positive functional value. This is continued, shrinking the interval of interest until it becomes smaller than some tolerance. Then the root is at the middle of the small interval remaining plus or minus half that tolerance.

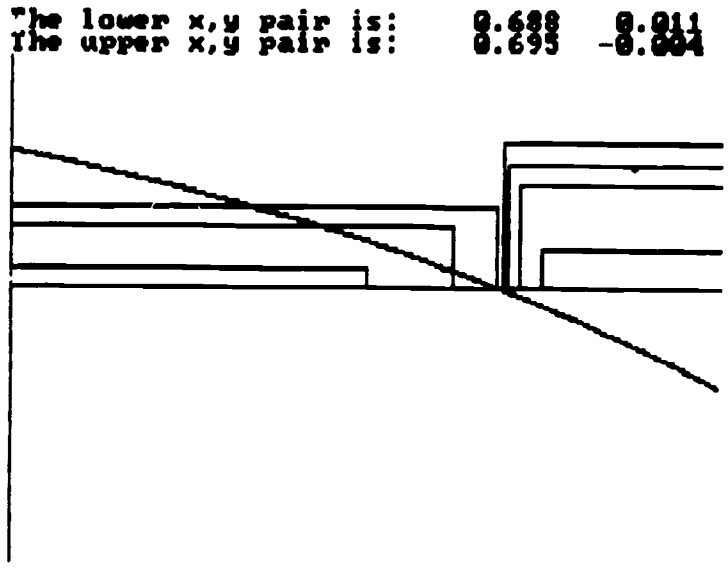
This program does the bisection graphically, showing the intervals on screen as higher and higher "walls" closing in on the root.

Listing of Program Bisection

```
program bisection(input,output);

{$igraphics.pas}  {These two statements include some graphics routines}
{$Igraphics.two}  {needed to run the program in Turbo Pascal on an IBM PC}
          {Listings of them can be found in the appendix.}

const xmin=0.0;
    ymin=-2.0;
    xlarge=1.0;
    ylarge=2.0;

var   x,lower,upper,mid,dot:real;
    signlow,i:integer;

function f(x:real):real;
{This function will be used in the bisection to
 compute values for finding the solution of f(x)=0}

begin {f}
  f:=2.0-exp(x)
end; {f}

function sign(y:real):integer;
  begin {sign}
    if (y=0.0) then sign:=0
    else if (y<0.0) then sign:=-1
    else sign:=1
  end; {sign}

begin {bisection}
  graphicson;
  plotaxes(xmin,xlarge,ymin,ylarge,1):
  x:=xmin;
  while(x<=xlarge) do begin
    plotgeneral(x,f(x),xmin,xlarge,
                ymin,ylarge,1);
    x:=x+0.0016
  end;
  dot:=0.0;      lower:=0.0;      upper:=1.0;
  signlow:=sign(f(0.0));
  while((upper-lower) > 0.01) do begin
    mid:=(upper+lower)/2.0;
    dot:=dot+0.15;
    if sign(f(mid))=0 then begin
      lower:=mid;
      upper:=mid;
      drawlinegeneral(xmin,dot,lower,dot,xmin,
                      xlarge,ymin,ylarge,1);
      drawlinegeneral(upper,0.0,upper,dot,xmin,
                      xlarge,ymin,ylarge,1)
      end {if}
```

```
          else if (sign(f(mid))=signlow) then begin
             lower:=mid;
             drawlinegeneral(xmin,dot,lower,dot,xmin,
                                        xlarge,ymin,ylarge,1);
             drawlinegeneral(lower,0.0,lower,dot,xmin,
                                        xlarge,ymin,ylarge,1)
             end
          else begin
             upper:=mid;
             drawlinegeneral(upper,dot,xlarge,dot,xmin,
                                        xlarge,ymin,ylarge,1);
             drawlinegeneral(upper,0.0,upper,dot,xmin,
                                        xlarge,ymin,ylarge,1)
          end
       end; {while}
       writeln('The lower x,y pair is: ',
                    lower:8:3,f(lower):8:3);
       writeln('The upper x,y pair is: ',
                    upper:8:3,f(upper):8:3);
       delay(5000);
       graphicsoff
    end. {bisect}
```

Sample Output



```
The lower x,y pair is:    0.688   0.011
The upper x,y pair is:    0.695  -0.004
```

BIBLIOGRAPHY

Frauenthal, James C., **Introduction to Population Modeling** Newton, Mass.: EDC/Project UMAP, 1978.

Hoffman, Dale T., **Monte Carlo: The Use of Random Digits to Simulate Experiments** Newton, Mass.: EDC/Project UMAP, 1978.

Milligan, W. Lloyd, "Wondrous Numbers and Other Divisors", **Creative Computing** 11 (April 1985), 99-101.

National Council of Teachers of Mathematics 1984 Yearbook, **Computers in Mathematics Education** edited by Viggo P. Hansen and Marilyn J. Zweng. Reston, Va: NCTM, 1984.

Wapner, Leonard M., "Modeling with Difference Equations: Two Examples", **Mathematics Teacher** 77 (Feb. 1984), 136-140.

Appendix

Graphics Procedures

AUTHOR: Lynn Ziegler

This appendix contains standard types and procedures used by graphics programs written for use with Turbo Pascal on the IBM PC. Similar programs were written for use with Turbo Pascal on Apple II computers with CP/M boards. Copies of either the IBM or Apple routines are available on request.

The following are the constants, types, and routines in the file "GRAPHICS.PAS" which is included in many of the above sample programs.

```
const
   xmax=319;
   ymax=199;
   pi = 3.1415926535;


type
   xvalue=0..xmax;
   yvalue=0..ymax;
   functionarray=array[0..xmax] of yvalue;


procedure cleargraphics;
   begin {cleargraphics}
      graphcolormode;
   end; {cleargraphics}


procedure graphicson;
{This procedure prepares the system for use of graphics.}

   begin {graphicson}
      graphcolormode;
   end; {graphicson}


procedure graphicsoff;
{This procedure restores normal text mode.}

   begin {graphicsoff}
      Textmode(bw80)
   end; {graphicsoff}
```

```
procedure plotpoint(x,y,color : integer);
{This procedure plots a point at coordinates x,y on the screen.  If color
 is 0 the background will be plotted.  Otherwise white will be plotted.}

   var xtemp : xvalue;
      ytemp : yvalue;

   begin {plotpoint}
      if ((0<=x) and (x<=xmax) and (0<=y) and (y<=ymax)) then begin
         xtemp:=abs(x) mod (xmax+1);
         ytemp:=ymax - (abs(y) mod (ymax+1));
         plot(xtemp,ytemp,color)
      end {if}
   end; {plotpoint}


procedure smoothplot(f : functionarray; color : integer);
{This procedure accepts an array f of xmax integers in the range 0 to ymax and
 plots them on the screen.  It assumes the integers represent a continuous
 function so it smooths vertical jumps as much as it can.}

   var i,j,mid : integer;

   begin {smoothplot}
      plot(0,f[0],color);
      for i:=1 to xmax do begin

         {Is there a jump where the current value is more than one dot
          above the previous value?  If so, plot the vertical values to
          fill in between the two points.}

         if f[i]>f[i-1]+1 then begin
            mid:=(f[i-1]+f[i]) div 2;
            for j:=f[i-1] to mid do plotpoint(i-1,j,color);
            for j:=f[i] downto (mid+1) do plotpoint(i,j,color)
            end {if}

         {Perhaps there is a jump down at least two dots?  If so, fill in
          the vertical values as above.}
         else if f[i]<f[i-1]-1 then begin
            mid:=(f[i-1]+f[i]) div 2;
            for j:=f[i-1] downto (mid+1) do plotpoint(i-1,j,color);
            for j:=f[i] to mid do plotpoint(i,j,color)
            end {elseif}

         {Maybe no jump occurs.  Then just plot the point.}
         else plotpoint(i,f[i],color)

      end {for}
   end; {smoothplot}
```

The following are the constants, types, and routines in the file "GRAPHICS.TWO" which is included in many of the above sample programs.

```
type
   arrayofreals=array[1..120] of real;


procedure drawline(x0,y0,x1,y1,color : integer);

{This procedure draws a line between the points (x0,y0) and (x1,y1) with shade
 color. The screen has coordinates 0 to ymax from bottom of screen to top of
 screen and 0 to xmax from left to right. The drawline uses full screen
 density and plots nice dense lines by checking slopes.}

   var
      x,y : integer;
      slope : real;

   begin {drawline}

      if (x0=x1) then {Vertical lines are plotted by simply changing y
                    and plotting the same x value (x0) at all points.}
         if (y0<y1) then
            for y:=y0 to y1 do plotpoint(x0,y,color)
         else
            for y:=y0 downto y1 do plotpoint(x0,y,color)
      else begin {Non vertical lines are plotted here.}
         slope:=(y1-y0)/(x1-x0);
         if (abs(slope)<=1.0) then {Small slopes are handled by varying
                              x and computing appropriate y values.}
            if (x0<x1) then
               for x:=x0 to x1 do
                  plotpoint(x,round(slope*(x-x0)+y0),color)
            else
               for x:=x0 downto x1 do
                  plotpoint(x,round(slope*(x-x0)+y0),color)
         else begin            {Large slopes are handled by thinking of x
                               as a function of y, varying y, and then
                               computing the proper x values.}
            slope:=1.0/slope;
            if (y0<y1) then
               for y:=y0 to y1 do
                  plotpoint(round(slope*(y-y0)+x0),y,color)
            else
               for y:=y0 downto y1 do
                  plotpoint(round(slope*(y-y0)+x0),y,color)
         end {if-then-else}
      end {if-then-else}
   end; {drawline}
```

```
procedure plotgeneral(x,y,xmin,xlarge,ymin,ylarge : real; color : integer);
{This procedure plots a point at position (x,y) in the coordinate system which
 places xmin as the leftmost x value, xlarge as the rightmost x value, ymin as
 the bottom y value, and ylarge as the top y value. It is plotted in shade
 color. It uses proportions and the specified values for standard screen
 coordinates (i.e., standard coordinates are x values 0 to xmax from left to
 right and 0 to ymax from bottom to top).}

   begin {plotgeneral}
      plotpoint(round((x-xmin)/(xlarge-xmin)*xmax),
            round((y-ymin)/(ylarge-ymin)*ymax), color)
   end; {plotgeneral}


procedure drawlinegeneral(x0,y0,x1,y1,xmin,xlarge,ymin,ylarge : real;
                                    color : integer);

{drawlinegeneral draws a line between (x0,y0) and (x1,y1) in the coordinate
 system running from xmin at left to xlarge at right and ymin at bottom and
 ylarge at top. It works by converting the points (x0,y0) and (x1,y1) to
 standard screen coordinates and then calling procedure drawline.}
   var
      xfactor,yfactor : real;

   begin {drawlinegeneral}
      xfactor:=xmax/(xlarge-xmin);
      yfactor:=ymax/(ylarge-ymin);
      drawline(round((x0-xmin)*xfactor),round((y0-ymin)*yfactor),
            round((x1-xmin)*xfactor),round((y1-ymin)*yfactor),color)
   end; {drawlinegeneral}


procedure drawcirclegeneral(xcenter,ycenter,radius  : real;
                     xmin,xlarge,ymin,ylarge : real;  color : integer);

{This procedure draws a circle centered at (xcenter,ycenter) of radius radius.
 The circle is in relative coordinate system xmin left to xlarge right and ymin
 bottom to ylarge top. The circle is drawn by using polar coordinates centered
 at (xcenter,ycenter). The relative number of points is proportional to the
 "normalized" radius and the number of points on the screens perimeter.}

   var
      theta,deltatheta : real;

   begin {drawcirclegeneral}
      theta:=0.0;
      deltatheta:=2.0*(xlarge-xmin+ylarge-ymin)/radius/(xmax+ymax);
      while (theta<=2.0*pi) do begin
         plotgeneral(xcenter+radius*cos(theta),ycenter+radius*sin(theta),
                  xmin,xlarge,ymin,ylarge,color);
         theta:=theta+deltatheta
      end {while}
   end; {drawcirclegeneral}
```

```
procedure plotaxes(xmin,xlarge,ymin,ylarge:real; color:integer);

{This procedure draws x and y axes through the origin (0,0) in the coordinate
 system defined by xmin left to xlarge right, ymin bottom to ylarge top in
 shade color.  It converts to standard screen coordinates to draw the line.}

begin {plotaxes}
   drawlinegeneral(0.0,ymin,0.0,ylarge,xmin,xlarge,ymin,ylarge,color);
   drawlinegeneral(xmin,0.0,xlarge,0.0,xmin,xlarge,ymin,ylarge,color)
end; {plotaxes}


procedure ConvertToPolar(x,y,xcenter,ycenter:real; var radius,theta:real);

{This procedure converts the point (x,y) to its representation in polar
 coordinates via the pair (radius,theta).  The polar coordinates are with
 reference to axes through (xcenter,ycenter).  Standard transformations are
 used for computing both radius and theta.}

begin {ConvertToPolar}
   radius:=sqrt(sqr(x-xcenter)+sqr(y-ycenter));
   if x=xcenter then
      if y>ycenter then theta:=pi/2
      else theta:=1.5*pi
   else if x>xcenter then theta:=arctan((y-ycenter)/(x-xcenter))
   else theta:=arctan((y-ycenter)/(x-xcenter))+pi
end; {ConvertToPolar}


procedure Rotate(var x,y:arrayofreals; theta:real;
             numberofpoints:integer; xcenter,ycenter:real);

{This procedure causes the numberofpoints points (x[i],y[i]) in the arrays x
 and y to be rotated by angle theta with respect to axes intersecting at
 (xcenter,ycenter).  Contents of arrays x and y will be changed.}

var
   i : integer;
   radius,theta0 : real;

begin {Rotate}
   for i:=1 to numberofpoints do begin
      ConvertToPolar(x[i],y[i],xcenter,ycenter,radius,theta0);
      x[i]:=xcenter+radius*cos(theta0+theta);
      y[i]:=ycenter+radius*sin(theta0+theta)
   end {for}
end; {Rotate}
```